A PROJECT REPORT

PATHFINDING ALGORITHM VISUALIZER

Submitted by

RAJESHWARAN. P . I (910719205018)

JOFFIN JOEL. J (910719205009)

In partial fulfillment for the award of the degree Of

BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY

K.L.N. COLLEGE OF INFORMATION TECHNOLOGY

POTTAPALAYAM.

ANNA UNIVERSITY: CHENNAI 600 025

JUNE 2022

BONAFIDE CERTIFICATE

Certified that this project report **"PATHFINDING ALGORITHM VISUALIZER"** is the bonafide work of **"RAJESHWARAN. P .I (910719205018),** *and* **JOFFIN JOEL. J (910719205009)"** who carried out the project work under my supervision.

SIGNATURE
Dr.E.S.VINOTHKUMAR,M.TECH.,pH.D
HEAD OF THE DEPARTMENT
Associate Professor
Information Technology
K.L.N. College of Information Technology,
Pottapalayam – 630612.
Sivagangai District,
TAMIL NADU

SIGNATURE
Mr.M.VENKATESH.,M.TECH
SUPERVISOR
Assistant Professor
Information Technology
K.L.N. College of Information Technology,
Pottapalayam – 630612.
Sivagangai District,
TAMIL NADU

Submitted for the project Viva-voce held at K.L.N. College of Information Technology, Pottapalayam on ……

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

Firstly, we would like to thank the Lord Almighty for his benevolence and blessings that has given us the strength to complete this project.

We express our sincere gratitude to our Principal **Dr.K.R.YOGANATHAN**, **B.E., M.E., Ph.D.** for providing us with plethora of resources, which enhances the quality of learning.

We express our sincere thanks to **Dr.E.S.VINOTHKUMAR M.TECH.,Ph.D.,Associate Professor** and Head of the Department (i/c), Department of Information Technology, for providing necessary facilities to carry out the project work.

We would like to express our deep sense of gratitude to our guide **Mr.M. VENKATESH, B.TECH., Assistant Professor**, Department of Information Technology, for his stimulating guidance, help and encouragement in the study, design and implementation of our project which enabled us to complete it successfully.

Also, we wish to thank all the faculty members of our department and our friends for rendering their support.

RAJESHWARAN. P.I

(910719205018)

JOFFIN JOEL. J

(910719205009)

ABSTRACT

Visualizations of algorithms contribute to improving computer science education. The process of teaching and learning of algorithms is often complex and hard to understand the problem. Visualization is a useful technique for learning in any computer science course. In this paper, an e-learning tool for shortest paths algorithms visualization is described. The Pathfinding algorithms are important because they are used in applications like google maps, satellite navigation systems, routing packets over the internet. The usage of pathfinding algorithms isn't just limited to navigation systems. The overarching idea can be applied to other applications as well. As a beginner’s step to algorithm and starting to understand how they work I decided to make this visualizer that shows in action how the pathfinding algorithm works in action. The understanding of this algorithm gives you the base idea of how navigation tools are implemented. In this visualizer, I have a grid page that has nodes that helped to pinpoint where to start and the endpoint. Also we can draw a line between the start node and the end node to make a barrier and see how our path will avoid it to reach the end node. To make this visualizer you need to be at least an intermediate level in front end programming (HTML, Nodes, CSS) and a good understanding of some pathfinding algorithm.

TABLE OF CONTENTS

ABSTRACT

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Pathfinding or pathing is the plotting, by a computer application, of the shortestroute between two points. It is a more practical variant on solving mazes.This field of research is based heavily on Dijkstra's algorithm for finding theshortest path on a weighted graph.

Pathfinding is closely related to the shortest path problem, within graph theory,which examines how to identify the path that best meets some criteria (shortest,cheapest, fastest, etc) between two points in a large network.At its core, a pathfinding method searches a graph by starting at one vertex and exploring adjacent nodes until the destination node is reached, generally with theintent of finding the cheapest route. Although graph searching methods such as a breadth-first search would find aroute if given enough time, other methods, which "explore" the graph, would tendto reach the destination sooner.An analogy would be a person walking across a room; rather than examiningevery possible route in advance, the person would generally walk in the directionof the destination and only deviate from the path to avoid an obstruction, andmake deviations as minor as possible.

CHAPTER 2

PROBLEM DEFINITION

Nowadays many students have good CGPA but they are rejected in coding interviews and don't have jobs. Because they have been struggling for a year to learn algorithms and still can't pass any technical interviews. Are algorithms hard to learn? Algorithms are probably one of the harder courses in your comp sci. degree, but it's totally doable. What makes it so difficult compared to other courses is how much intuition is involved in designing/analyzing algorithms.

Pathfinding algorithms are important because they are used in applications like google maps, satellite navigation systems, routing packets over the internet. The usage of pathfinding algorithms isn't just limited to navigation systems. The overarching idea can be applied to other applications as well. But many of the students just study the algorithm but not having an idea about the algorithm. This visualizer help students to visualize and get an idea about how algorithms are work

and where we use this algorithm. Many of students easily know it and crack the interviews efficiently.

CHAPTER 3

PROJECT GOALS AND OBJECTIVES

The most important factor for this project to visualize algorithm to easily understand by students.

In this section, the overall working of the project has been described. How the project started and how the project works and how the various phases of project were carried out and the challenges faced at each level. What does the project do? At its core, a pathfinding algorithm seeks to find the shortest path between two points. This project visualizes various pathfinding algorithms in action, and more! All of the algorithms on this project are adapted for a 2D grid, where 90 degree turns have a "cost" of 1 and movements from a node to another have a "cost" of 1.

Picking an Algorithm:
Choose an algorithm from the "Algorithms" drop-down menu. Note that some algorithms are unweighted, while others are weighted. Unweighted algorithms do not take turns or weight nodes into account, whereas weighted ones do.

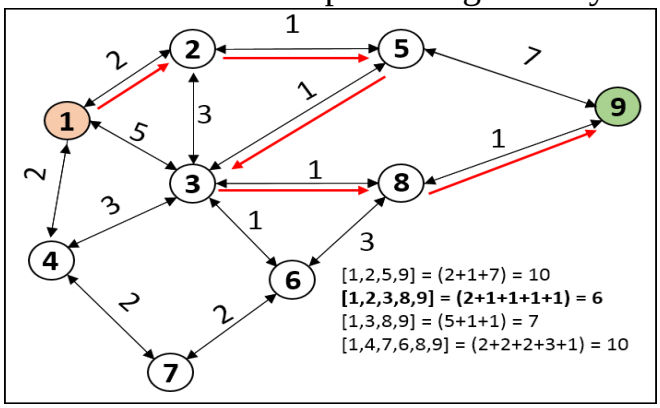
Objectives of the project:
- It can be used as a E learning tool to understand Algorithms.
- It is used in finding Shortest Path.
- It is used in the telephone network.
- It is used in IP routing to find Open shortest Path First.
- It is used in geographical Maps to find locations of Map which refers to vertices of graph.
- We can make a GPS system which will guide you to the locations.
- Search engine crawlers are used BFS to build index. Starting from source page, it finds all links in it to get new pages.
- In peer-to-peer network like bit-torrent, BFS is used to find all neighbor nodes.
- As users of wireless technology, people demand high data rates beyond GigaBytes per second for Voice, Video and other applications. There are many standards to achieve data rates beyond GB/s. One of the standards is MIMO(Multi input Multi output).MIMO employs K-best Algorithm(which isa Breadth-First Search
algorithm) to find the shortest partial euclidian distances.

CHAPTER 4

PATHFIND ALGORITHMS

4.1 Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a weighted graph. We'll use the new addEdge and addDirectedEdge methods to add weights to the edges when creating a graph. Let us look at how this algorithm works −

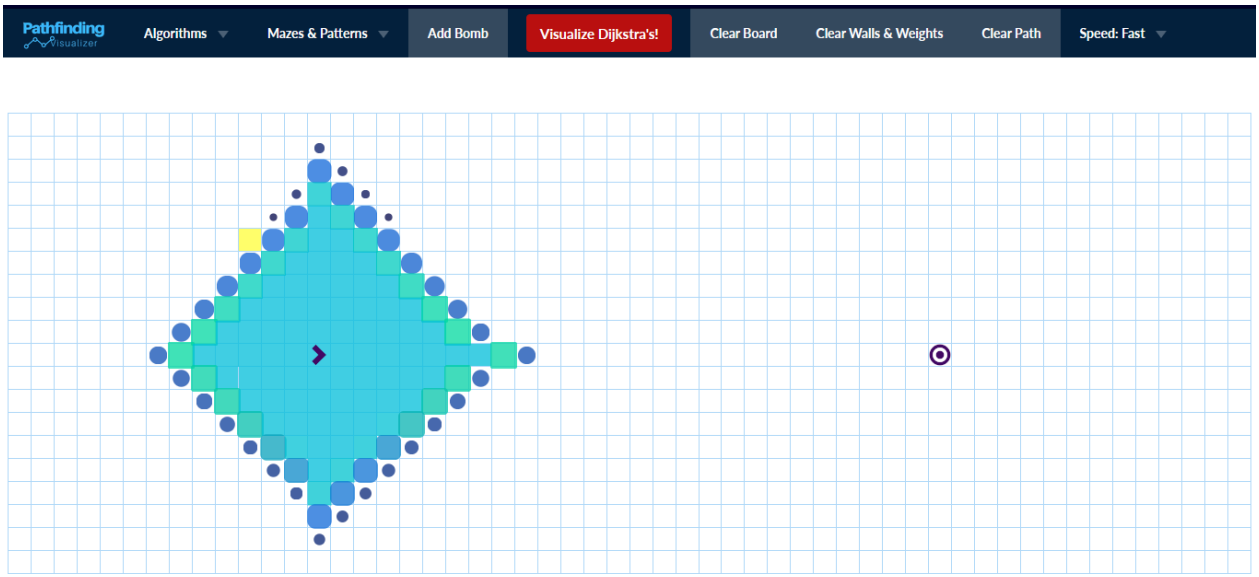
- Create a distance collection and set all vertices distances as infinity except the source node.
- Enqueue the source node in a min-priority queue with priority 0 as the distance is 0.
- Start a loop till the priority queue is empty and dequeue the node with the minimum distance from it.
- Update the distances of the connected nodes to the popped node if "current node distance + edge weight < next node distance", then push the node with the new distance to the queue.
- What this algorithm basically does is assumes all nodes are at an infinite distance from the source. Then it starts taking the edges in consideration and keeps track of distances of nodes from the source updating the same if it finds a lower cost path along the way.



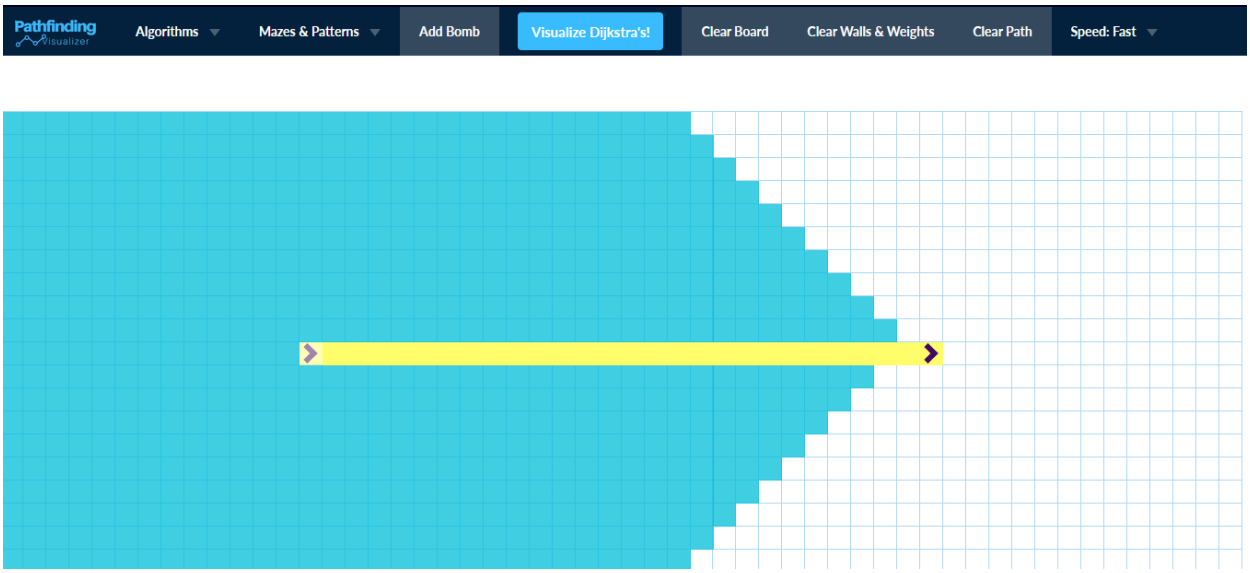
Let us look at this implementation in code (Javascipt) –

```
djikstraAlgorithm(startNode) {
    let distances = {};

    // Stores the reference to previous nodes
    let prev = {};
    let pq = new PriorityQueue(this.nodes.length * this.nodes.l
ength);

    // Set distances to all nodes to be infinite except startNo
de
    distances[startNode] = 0;
    pq.enqueue(startNode, 0);
    this.nodes.forEach(node => {
        if (node !== startNode) distances[node] = Infinity;
        prev[node] = null;
    });

    while (!pq.isEmpty()) {
        let minNode = pq.dequeue();
        let currNode = minNode.data;
        let weight = minNode.priority;
        this.edges[currNode].forEach(neighbor => {
            let alt = distances[currNode] + neighbor.weight;
            if (alt < distances[neighbor.node]) {
                distances[neighbor.node] = alt;
                prev[neighbor.node] = currNode;
                pq.enqueue(neighbor.node, distances[neighbor.node]
);
            }
        });
    }
    return distances;
 }
```

OUTPUT:



(a) **finding shortest path**



(b) **The shortest path**

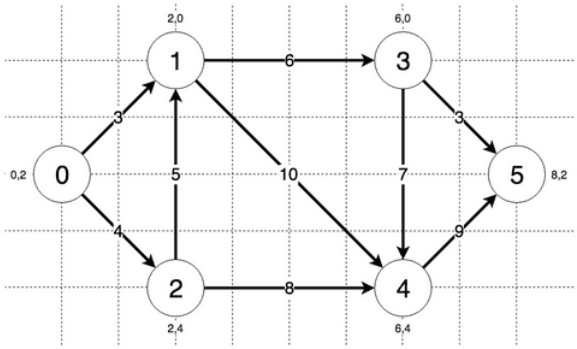
.

4.2 A Star Algorithm is an algorithm used to solve the shortest path problem in a graph. This means that given a number of nodes and the edges between them as well as the "length" of the edges (referred to as "weight") and a heuristic (more on that later), the A* algorithm finds the shortest path from the specified start node to all other nodes. Nodes are sometimes referred to as vertices (plural of vertex) - here, we'll call them nodes.

In more detail, this leads to the following Steps:

1. Initialize the distance to the starting node as 0 and the distances to all other nodes as infinite
2. Initialize the priority to the starting node as the straight-line distance to the goal and the priorities of all other nodes as infinite
3. Set all nodes to "unvisited"
4. While we haven't visited all nodes and haven't found the goal node:
    1. Find the node with currently lowest priority (for the first pass, this will be the source node itself)
    2. If it's the goal node, return its distance
    3. For all nodes next to it that we haven't visited yet, check if the currently smallest distance to that neighbor is bigger than if we were to go via the current node
    4. If it is, update the smallest distance of that neighbor to be the distance from the source to the current node plus the distance from the current node to that neighbor, and update its priority to be the distance plus its straight-line distance to the goal node.

Consider the following graph:

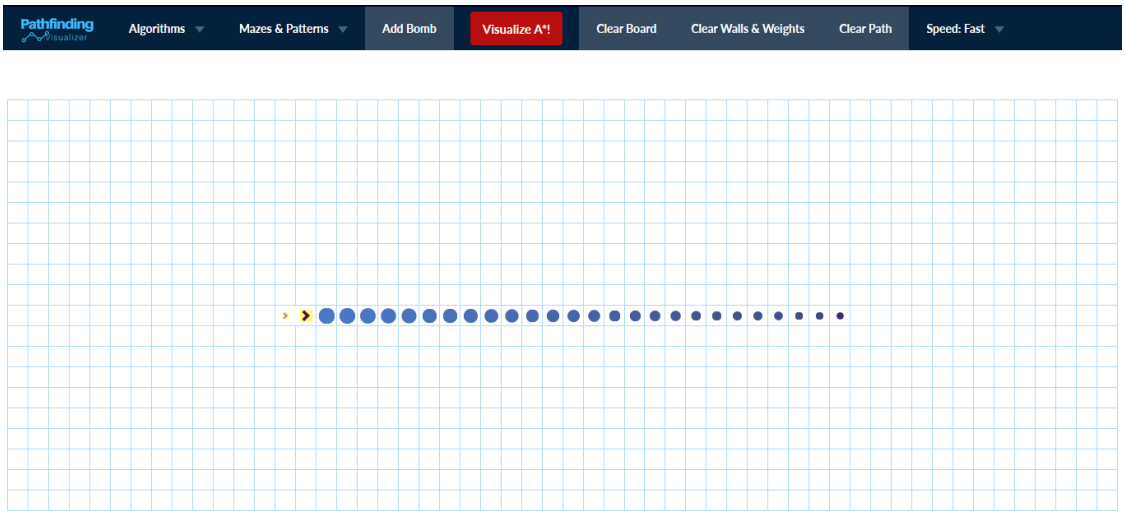


```
const aStar = function (graph, heuristic, start, goal) {
  var distances = [];
  for (var i = 0; i < graph.length; i++) distances[i] = Number.MAX_VALUE;
  distances[start] = 0;
  var priorities = [];
  for (var i = 0; i < graph.length; i++) priorities[i] = Number.MAX_VALUE;
  priorities[start] = heuristic[start][goal];
  var visited = [];
  while (true) {
    var lowestPriority = Number.MAX_VALUE;
    var lowestPriorityIndex = -1;
    for (var i = 0; i < priorities.length; i++) {
      if (priorities[i] < lowestPriority && !visited[i]) {
        lowestPriority = priorities[i];
        lowestPriorityIndex = i;
      }
    }
    if (lowestPriorityIndex === -1) {
      return -1;
    } else if (lowestPriorityIndex === goal) {
      return distances[lowestPriorityIndex];
    }
    for (var i = 0; i < graph[lowestPriorityIndex].length; i++) {
      if (graph[lowestPriorityIndex][i] !== 0 && !visited[i]) {
        if (distances[lowestPriorityIndex] + graph[lowestPriorityIndex][i] < dist
ances[i]) {
          distances[i] = distances[lowestPriorityIndex] + graph[lowestPriorityIn
dex][i];
          priorities[i] = distances[i] + heuristic[i][goal];          }
      }
    }
    visited[lowestPriorityIndex] = true;
  }
};
module.exports = {aStar};
```

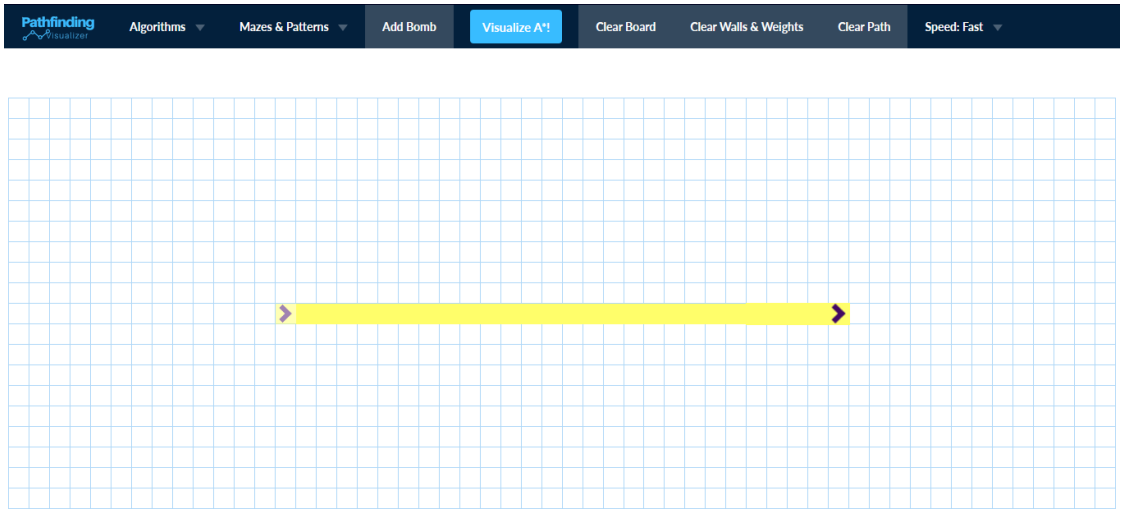
The steps the algorithm performs on this graph if given node 0 as a starting point and node 5 as the goal, in order, are:

1. Visiting node 0 with currently lowest priority of 8.0
2. Updating distance of node 1 to 3 and priority to 9.32455532033676
3. Updating distance of node 2 to 4 and priority to 10.32455532033676
4. Visiting node 1 with currently lowest priority of 9.32455532033676
5. Updating distance of node 3 to 9 and priority to 11.82842712474619
6. Updating distance of node 4 to 13 and priority to 15.82842712474619
7. Visiting node 2 with currently lowest priority of 10.32455532033676
8. Visiting node 3 with currently lowest priority of 11.82842712474619
9. Updating distance of node 5 to 12 and priority to 12.0
10. Goal node found!

OUTPUT:



(a) **finding shortest path**



(b) **The shortest path**

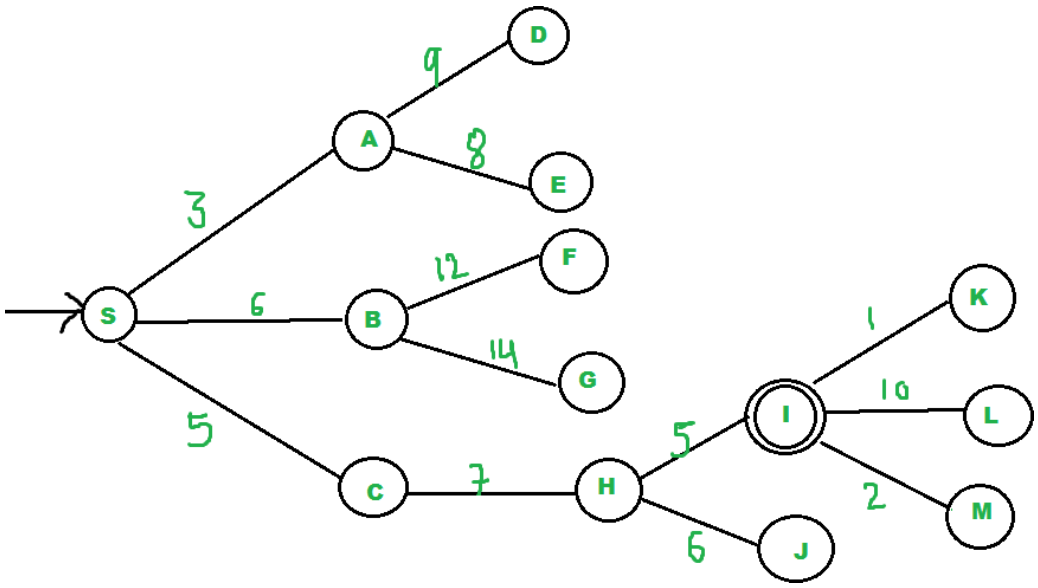
4.3 GREEDY BEST-FIRST SEARCH - In BFS and DFS, when we are at a node, we can consider any of the adjacent as next node. So both BFS and DFS blindly explore paths without considering any cost function. The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

We use a priority queue to store costs of nodes. So the implementation is a variation of BFS, we just need to change Queue to PriorityQueue.

PSEUDO CODE:

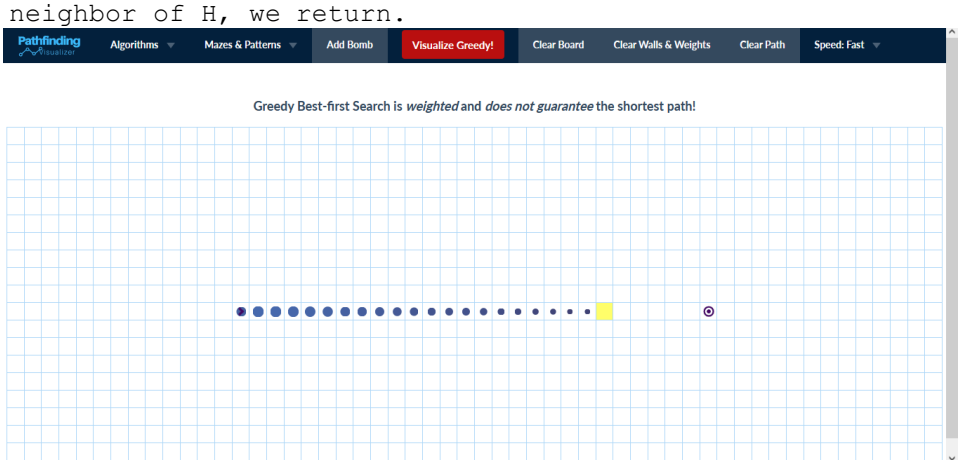
```
Best-First-Search(Grah g, Node start)
    1) Create an empty PriorityQueue
       PriorityQueue pq;
    2) Insert "start" in pq.
       pq.insert(start)
    3) Until PriorityQueue is empty
          u = PriorityQueue.DeleteMin
          If u is the goal
             Exit
          Else
             Foreach neighbor v of u
                If v "Unvisited"
                    Mark v "Visited"
                    pq.insert(v)
             Mark u "Examined"
End procedure
```

Let us consider the below example.



OUTPUT:

```
We start from source "S" and search for
goal "I" using given costs and Best
First search.

pq initially contains S
We remove s from and process unvisited
neighbors of S to pq.
pq now contains {A, C, B} (C is put
before B because C has lesser cost)

We remove A from pq and process unvisited
neighbors of A to pq.
pq now contains {C, B, E, D}

We remove C from pq and process unvisited
neighbors of C to pq.
pq now contains {B, H, E, D}

We remove B from pq and process unvisited
neighbors of B to pq.
pq now contains {H, E, D, F, G}

We remove H from pq.  Since our goal
"I" is a neighbor of H, we return.
```



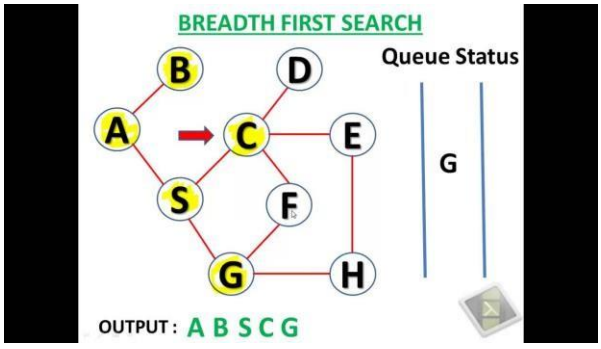
(a) finding shortest path

(b) The shortest path

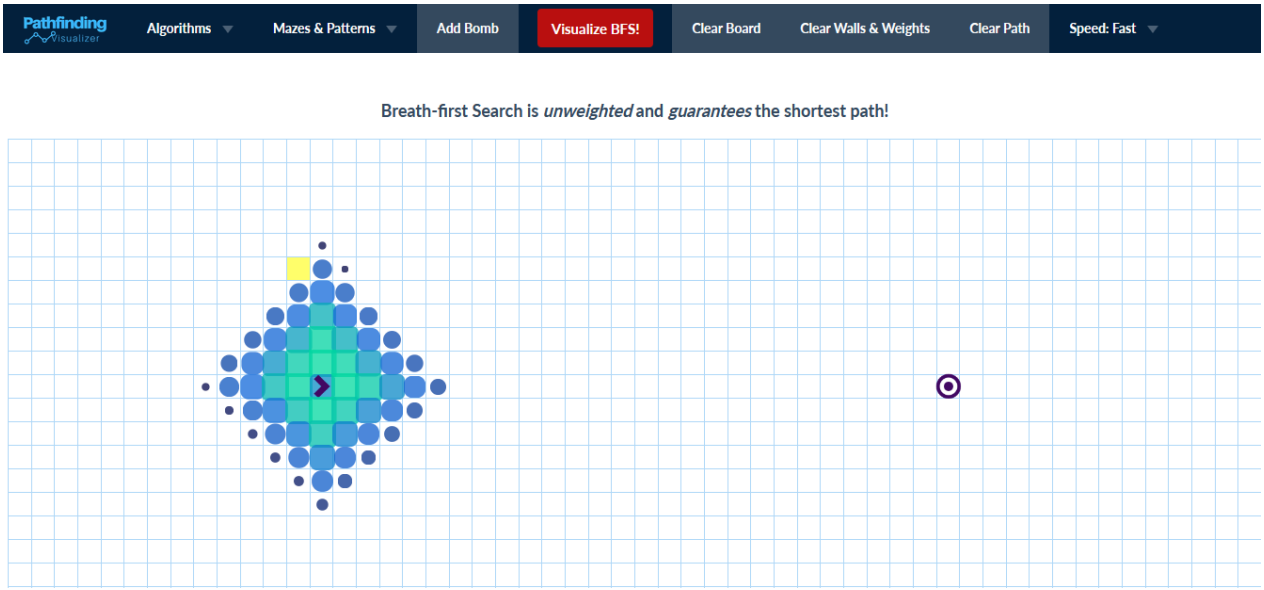
4.4 BREADTH-FIRST SEARCH for a graph is similar to Breadth First Traversal of a tree .The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.

For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Breadth First Traversal of the following graph is 2, 0, 3, 1.

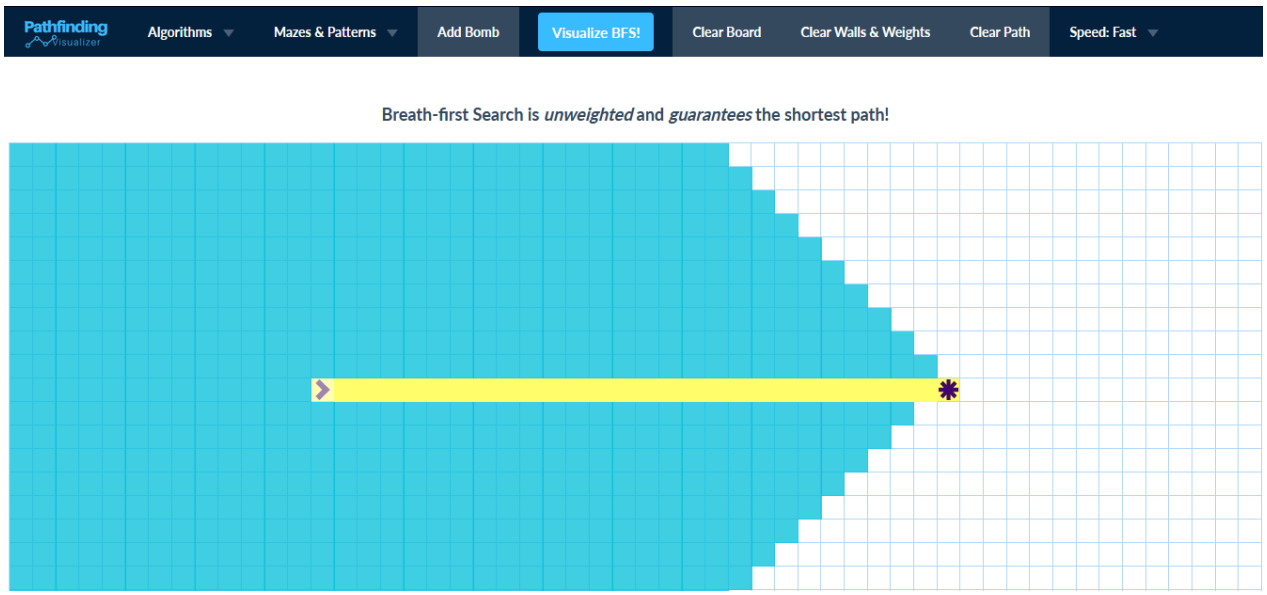
```
BFS(node) {
    // Create a Queue and add our initial node in it
    let q = new Queue(this.nodes.length);
    let explored = new Set();
    q.enqueue(node);
    // Mark the first node as explored explored.
    add(node);
    // We'll continue till our queue gets empty
    while (!q.isEmpty()) {
       let t = q.dequeue();

       this.edges[t].filter(n => !explored.has(n)).forEach(n => {          ex
plored.add(n);
          q.enqueue(n);
       });
    }
 }
```



OUTPUT:



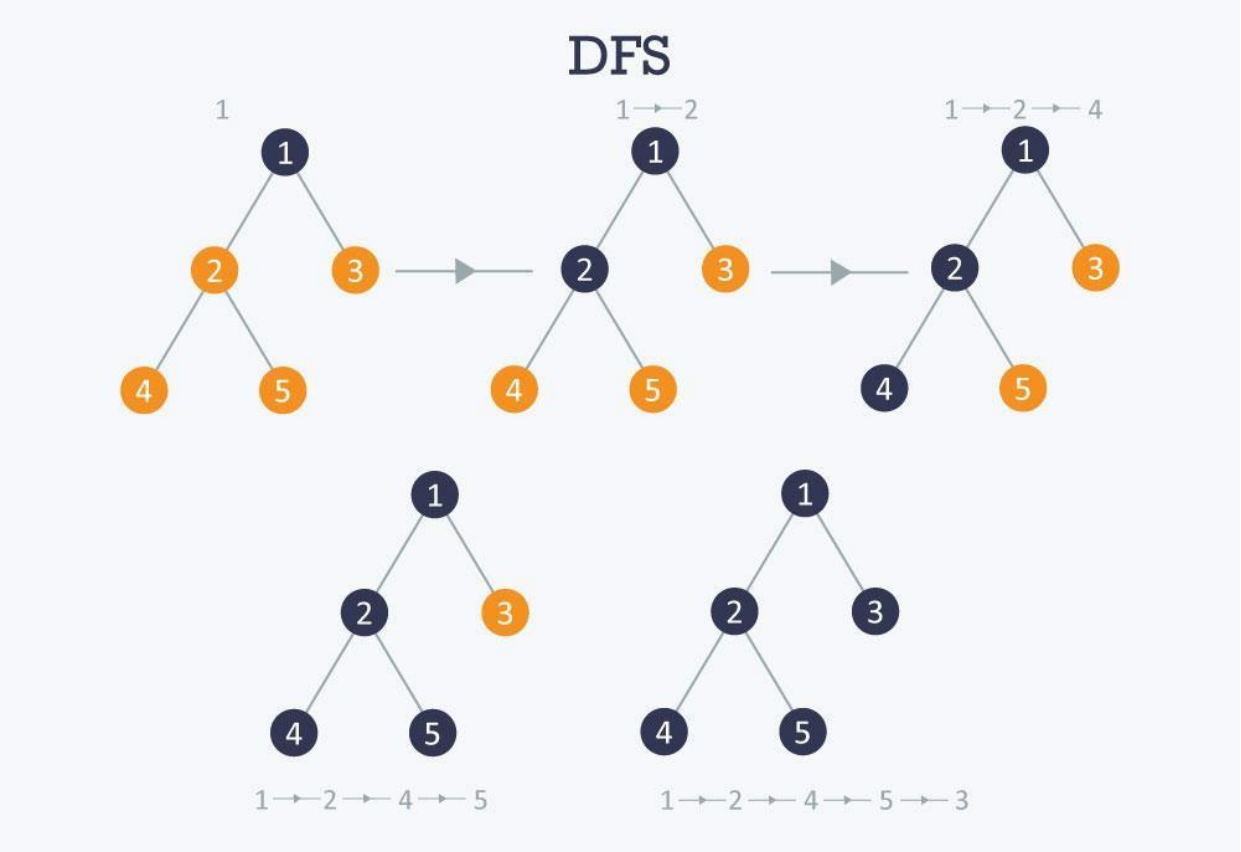
(a) **finding shortest path**



(b) **The shortest path**

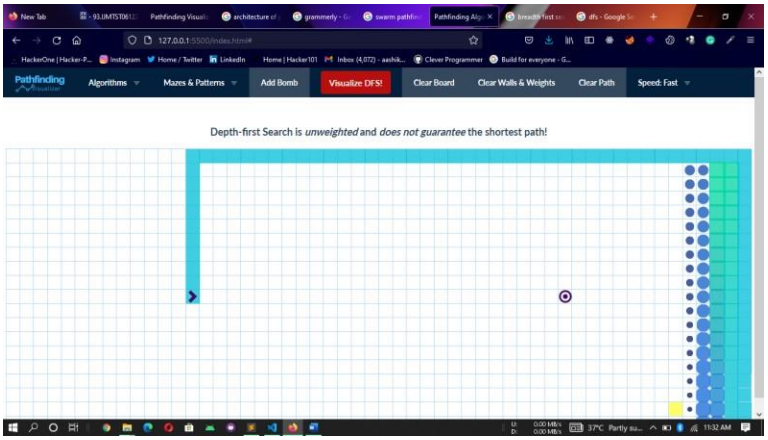
4.5 DEPTH-FIRST SEARCH - for a graph is similar to Depth First Search Tree The only catch here is, unlike trees, graphs may contain cycles, a node may be visited twice. To avoid processing a node more than once, use a boolean visited array.

- **Approach:** Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.
- **Algorithm:**
    1. Create a recursive function that takes the index of the node and a visited array.
    2. Mark the current node as visited and print the node.
    3. Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

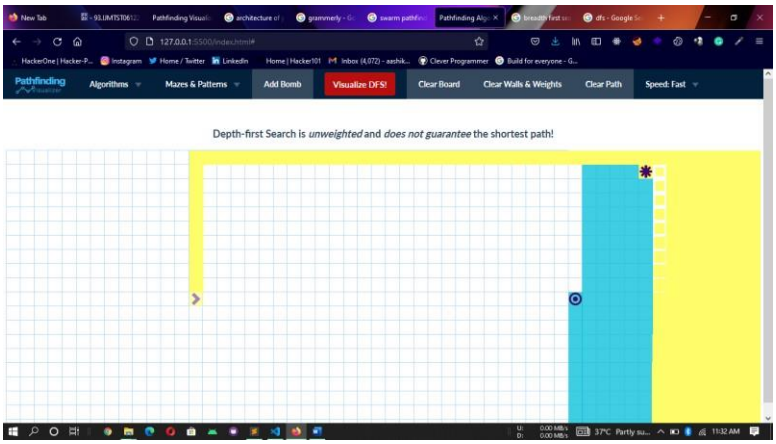


```
DFS(node) {
    // Create a Stack and add our initial node in it
    let s = new Stack(this.nodes.length);
    let explored = new Set();
    s.push(node);

    // Mark the first node as explored
    explored.add(node);

    // We'll continue till our Stack gets empty
    while (!s.isEmpty()) {
        let t = s.pop();

    // Log every element that comes out of the Stack
        console.log(t);

    // 1. In the edges object, we search for nodes this node is
 directly connected to.
    // 2. We filter out the nodes that have already been explor
ed.
    // 3. Then we mark each unexplored node as explored and pus
h it to the Stack.
    this.edges[t]
    .filter(n => !explored.has(n))
    .forEach(n => {
        explored.add(n);
        s.push(n);
        });
    }
 }
```

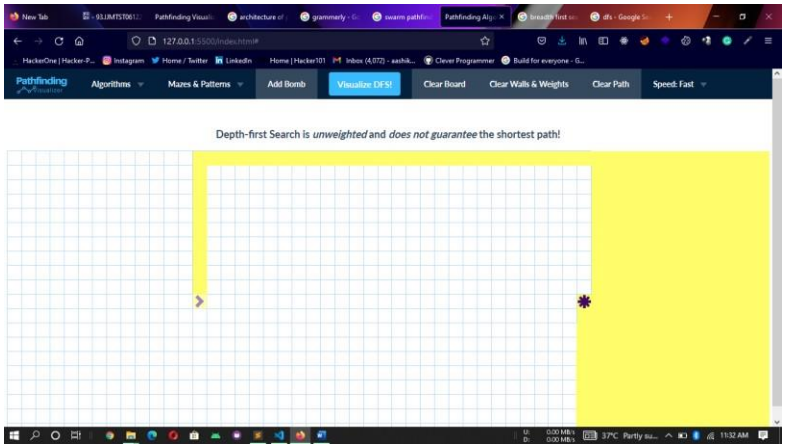
OUTPUT:



(a) finding shortest path - I



(b) finding shortest path - II

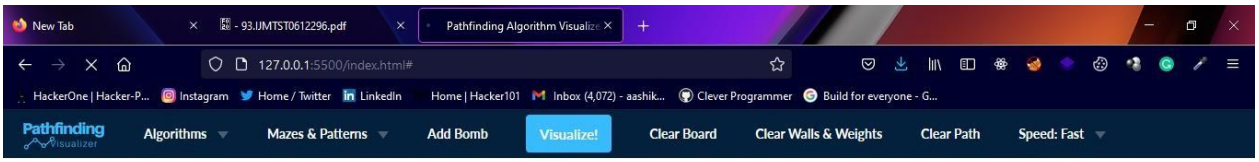


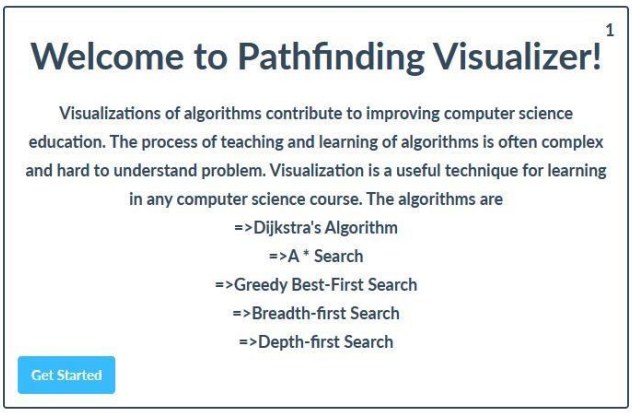
(c) The shortest path - II

CHAPTER 5

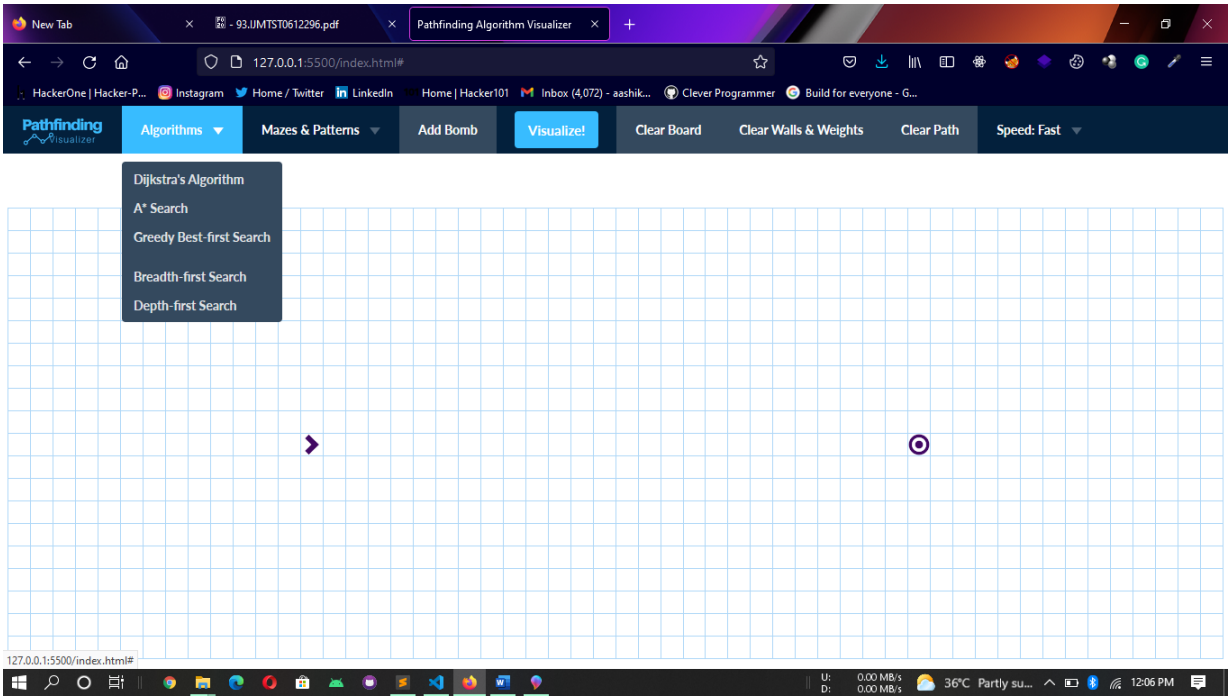
RESULTS

- FirstlyClick on Get started.

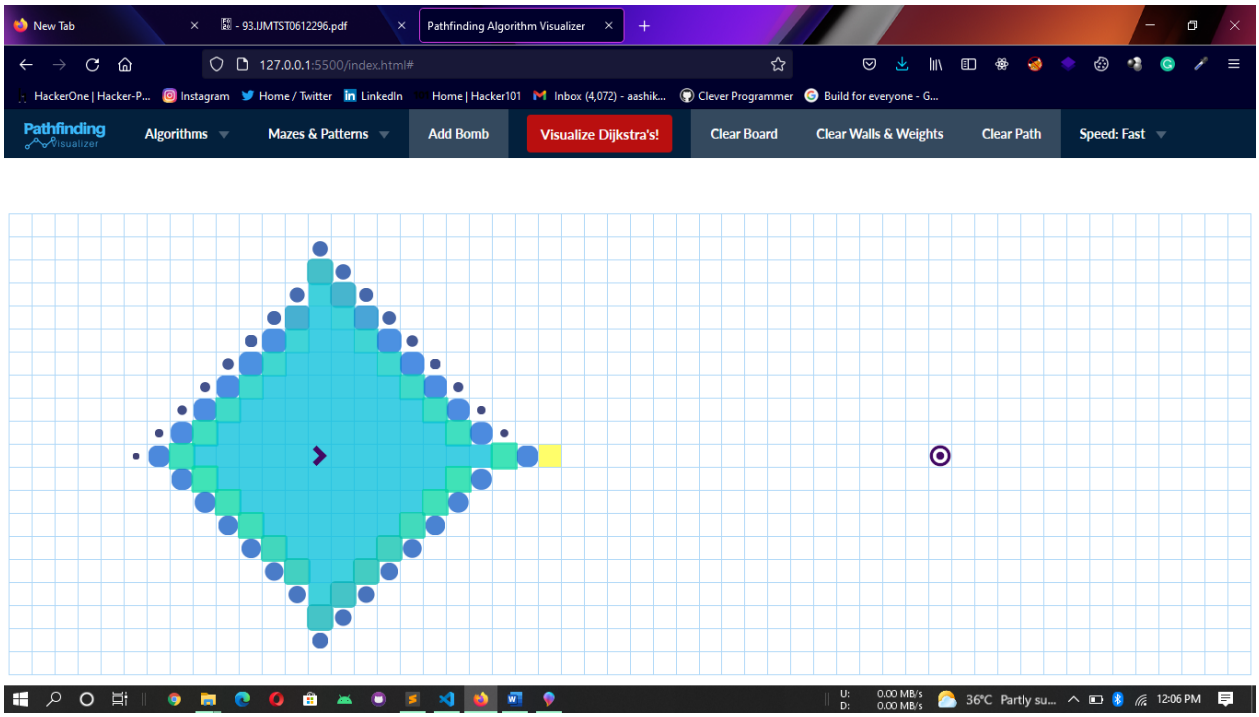




- Select Algorithm in Algorithms drop down box.



- Click Visualize button to start.



CHAPTER 6

PERFORMANCE ANALYSIS AND COMPARISION

TABLE:

ALGORTITHM	TIME(s:ms)
Dijkstra's Algorithm	21:26
A Star Algorithm	1:45
Greedy Best-First Search	1:34
Breadth-First Search	21:85
Depth-First Search	7:14

CHAPTER 7

SAMPLE SOURCE CODE

<!--index.html-->

```
<html>
  <head>
    <title>Pathfinding Algorithm Visualizer</title>
    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css
/bootstrap.min.css"
    />
    <link id="cssTheme" rel="stylesheet" href="public/styling/c
ssBasic.css" />
    <link
      rel="shortcut icon"
      type="image/png"
      href="public/styling/c_icon.png"
    />
  </head>
  <body>
    <div id="navbarDiv">
      <nav class="navbar navbar-inverse" style="background-
color: #03203c">
        <div class="container-fluid">
          <div class="navbar-header">
            <a id="refreshButton" class="navbar-brand" href="#"
              ><img
                src="./public/styling/logo.png"
                alt=""
                width="90px"
                height="30px"
            /></a>
          </div>
          <ul class="nav navbar-nav" style="background-
color: #03203c">
```

```
                <li class="dropdown">
                  <a class="dropdown-toggle" data-
toggle="dropdown"  href="#"
                    >Algorithms <span class="caret"></span
                  ></a>
                  <ul  class="dropdown-menu">
                    <li id="startButtonDijkstra">
                      <a  href="#">Dijkstra's Algorithm</a>
                    </li>
                    <li id="startButtonAStar2"><a href="#">A* Searc
h</a></li>
                    <li id="startButtonGreedy">
                      <a href="#">Greedy Best-first Search</a>
                    </li>
                    <li  id="startButtonAStar"><a  href="#"></a></li>
                    <li id="startButtonAStar3">
                      <a  href="#"></a>
                    </li>
                    <li id="startButtonBidirectional">
                      <a  href="#"></a>
                    </li>
                    <li  id="startButtonBFS">
                      <a  href="#">Breadth-first Search</a>
                    </li>
                    <li  id="startButtonDFS"><a  href="#">Depth-
first Search</a></li>
                  </ul>
                </li>
                <li class="dropdown">
                  <a class="dropdown-toggle" data-
toggle="dropdown"  href="#"
                    >Mazes &amp;amp; Patterns <span  class="caret"></spa
n
                  ></a>
                  <ul  class="dropdown-menu">
                    <li  id="startButtonCreateMazeTwo">
                      <a href="#">Recursive  Division</a>
```

```
                    </li>
                    <li id="startButtonCreateMazeThree">
                      <a href="#">Recursive Division (vertical skew
)</a>
                    </li>
                    <li id="startButtonCreateMazeFour">
                      <a href="#">Recursive Division (horizontal sk
ew)</a>
                    </li>
                    <li id="startButtonCreateMazeOne">
                      <a href="#">Basic Random Maze</a>
                    </li>
                    <li id="startButtonCreateMazeWeights">
                      <a href="#">Basic Weight Maze</a>
                    </li>
                    <li id="startStairDemonstration">
                      <a href="#">Simple Stair Pattern</a>
                    </li>
                  </ul>
                </li>
                <li id="startButtonAddObject"><a href="#">Add Bomb<
/a></li>
                <li id="startButtonStart">
                  <button
                    id="actualStartButton"
                    class="btn btn-default navbar-btn"
                    type="button"
                  >
                    Visualize!
                  </button>
                </li>
                <li id="startButtonClearBoard"><a href="#">Clear Bo
ard</a></li>
                <li id="startButtonClearWalls">
                  <a href="#">Clear Walls &amp;amp; Weights</a>
                </li>
```

```
                <li id="startButtonClearPath"><a href="#">Clear Path</a></li>
                <li class="dropdown">
                  <a
                    id="adjustSpeed"
                    class="dropdown-toggle"
                    data-toggle="dropdown"
                    href="#"
                    >Speed: Fast <span class="caret"></span
                  ></a>
                  <ul class="dropdown-menu">
                    <li id="adjustFast"><a href="#">Fast</a></li>
                    <li id="adjustAverage"><a href="#">Average</a></li>
                    <li id="adjustSlow"><a href="#">Slow</a></li>
                  </ul>
                </li>
              </ul>
            </div>
          </nav>
        </div>
        <div id="tutorial">
          <h3>Welcome to Pathfinding Visualizer!</h3>
          <p>
            Visualizations of algorithms contribute to improving computer science
            education. The process of teaching and learning of algorithms is often
            complex and hard to understand problem. Visualization is a useful
            technique for learning in any computer science course. The algorithms
            are <br />=>Dijkstra's Algorithm <br />=>A * Search <br />=>Greedy
            Best-First Search <br />=>Breadth-first Search <br />=>Depth-first
            Search
```

```
        </p>
        <div id="tutorialCounter">1</div>

        <button
          id="skipButton"
          class="btn btn-default navbar-btn"
          style="background-color: #38bcff"
          type="button"
        >
          Get Started
        </button>
      </div>
      <div id="mainGrid">
        <div id="mainText"></div>
        <div id="algorithmDescriptor"></div>
        <table id="board" />
      </div>
    </body>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1
.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/
js/bootstrap.min.js"></script>
    <script src="public/browser/bundle.js"></script>
</html>
```

Full Source code:

https://github.com/aashikm015/PathfindingAlgorithmVisualizer.git

CHAPTER 8

CONCLUSION

With the completion of this project, we have successfully achieved our objective of our project is to embed Graph Path Finding with Visualization and Comparing their performance.

As is the case with most other teaching areas, there has been a significant gap between the theory and practical understanding of algorithms realization. This is true also for shortest paths algorithms and in particular for Dijkstra algorithm. The main goal of the project is to use it from operations research educators and students for teaching and studying the existing known combinatorial graph algorithms.

The main idea of the system is to provide an integrated educational environment for both instructors and students to facilitate the learning process in efficient way.

To conclude, we have learnt a lot of things working under this project. We are also thankful to our mentor and supervisor for their efforts in the learning process.

CHAPTER 9

REFERENCES

1. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

2. https://medium.com/swlh/pathfinding-algorithms-6c0d4febe8fd#:~:text=What%20are%20Pathfinding%20Algorithms%3F,based%20on%20some%20predefined%20criteria.

3. https://en.wikipedia.org/wiki/Pathfinding

4. https://www.w3schools.com/js/

5.https://www.meta-chart.com/histogram 6.https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

7. https://www.geeksforgeeks.org/a-search-algorithm/

8. https://www.researchgate.net/publication/282488307_Pathfinding_Algorithm_Efficiency_Analysis_in_2D_Grid

9. http://tesi.fabio.web.cs.unibo.it/twiki/pub/Tesi/DocumentiRitenuti Utili/p80-ardito.pdf

10. Ahuja R. K., Magnanti, T. L. & Orlin, J. B.(1993). Network Flows: Theory,Algorithms and Applications. Englewood Cliffs, NJ: Prentice Hall.

11. Dijkstra, E. W. (1959). A Note on Two Problems in Connection with Graphs. Numeriche Mathematik, 1, 269-271.

12. Fouh E., Akbar M. & Shaffer C. A. (2012). The Role of Visualization in Computer Science Education. Computers in the Schools, 29(1-2), 95-117.

13. Roles J.A. & ElAarag H. (2013). A Smoothest Path algorithm and its visualization tool. Southeastcon, In Proc. of IEEE, DOI: 10.1109/SECON.2013.6567453.

14. Paramythis A., Loidl S., Mühlbacher J. R., & Sonntag M. (2005). A Framework for Uniformly Visualizing and Interacting with Algorithms. In Montgomerie, T.C., & Parker E-Learning, J.R. (Eds.), In Proc. IASTED Conf. on Education and Technology (ICET 2005), Calgary, Alberta, Canada, 2-6 July 2005, pp. 28- 33.